The FeriteDoc Guide 1.0 - Documenting Ferite Applications

May 2, 2005

Contents

1 Introduction

1.1 feritedoc

feritedoc is a documentation tool for ferite modules and scripts. It allows you, the developer, to keep your documentation tightly integrated with your code. The html template produces nice looking pages with an integrated search ability.

1.2 Background

Chris Ross, the author of ferite, wrote feritedoc as a handy tool for documenting ferite scripts. feritedoc has similarities to other documentation programs, but is actually a ferite script, so it is installed with the ferite distribution.

1.3 Assumptions

This document is written for the experienced ferite developer. It assumes that you have ferite installed, know how to create scripts and are interested in adding in-line documentation to your scripts.

1.4 The Obligatory Whirlwind Tour

feritedoc is pretty easy to use. You tell it what script you're going to use as input, what template you're going to use for the output and where you want to create those files. You can leave out the location if you want the files to be created in your working directory.

feritedoc reads the input script and looks for document blocks. These are C style comments that start with "/**" and end with "*/." Here's a short example.

```
/**
 * @namespace myTest
 * @brief Documentation for my test scripts.
 */
namespace myTest {

/**
 * @function foobar
 * @brief This function converts a string to foobiness.
 * @declaration function foobar( string target )
 * @param string target The string to be foobied.
 * @description The target parameter is not updated, instead
 * a foobie copy is created and returned.
 * @return A copy of the target string that has been foobied.
 * @example <nl />
 * <code>Console.println("foobar(${target}) => ${foobar(target)}");</code>
 */
function foobar( string target ) {
  return "foo-" + target + "-bar";
}

/**
 * @end
 */
}
```

When run with the following command line,

```
$ feritedoc --prefix docs/ --template html --file myTest.fe
```

it produces the following output files:

```
docs/myTest.html

docs/jse_sites.js
docs/list.html
docs/main.html

docs/back.png
docs/class.png
docs/empty.png
docs/function.png
docs/index.html
docs/jse_form.js
docs/jse_search.js
docs/namespace.png
docs/results.html
docs/style.css
docs/var.png
```

The first file (myTest.html) is created specifically for the namespace. It gets its name from the @namespace specified (in this case, myTest). Each @namespace in the script will have a separate html file.

The next three files (jse_sites.js, list.html and main.html) are standard files that are customized to point to the namespace page.

The remaining files are standard boilerplate. They are copied from the template directory without modification.

To view the documentation, simply load the index.html file into your favorite browser. You should see something similar to the following:

It is not difficult to customize the output. However, you do need to be *very* familiar with the internals of the html template.

2 Running

2.1 Overview

This section desribes how to run feritedoc from the command line. The most common command line options are mentioned. See The section On Command Line Options for full details on all options.

2.2 Usage

feritedoc has a very easy to use command line interface. For most situations, only three parameters will be used: –template, –prefix and –file.

The –template option specifies which template to use for the generated documents. It is the only required option.

The –file option specifies the script name to document. It is optional. You could take advantage of that to re-create all of the boilerplate files after an upgrade. But, it doesn't make much sense to generate documentation without specifying a file to process, right?

The –prefix option specifies a string to prepend to all output file names. It is optional. You can use this option to specify a simple name or a complete path. If you're specifying a path, be sure to include the trailing slash ("/").

Other command line options are detailed in **command line options** .

The simplest usage is

```
    $ feritedoc --template html --file myTest.fe
```

The –template option specifies that we'll be using the html template. That's the template used in the ferite standard documentation. It will create one file per "group" in the script (grouping is covered in **markup tagging**). The other template type is "text," which we won't mention except to mention that we won't mention it.

3 Running

All templates use a common processor (the ”parser”) that extracts in-line documentation from the script files. The parser examines each script looking for documentation blocks. When it finds a documentation block, it scans the block looking for markup tags. All templates use the same markup tags, they only differ in how they format the output.

It is important to note that the html template requires that you provide at least one grouping tag in your script (grouping tags are covered in The Next Section). If you don’t, feritedoc will either create an empty set of documents or display the following error message:

```
Error:  Trying to access variable ’doctype’ in object ’b’ which is
null
```

3.1 Documentation Blocks

All of the markup tags are embedded in specially formatted C style comments. A documentation block starts with ”/**” and ends with ”*/.” The block looks like this:

```
/**
 * This is a C style documentation block.
 * Note that there are two asterisks on the first line.
 */
```

All comments that have this format will be scanned by the parser.

There are some limitations to the parser. For example, the following

```
string foolMeOnce = "/**";
string foolMeTwice = " * @namespace";
```

can cause it to blow up. The parser doesn’t understand that the comment is quoted, so it isn’t really a comment.

3.2 Three Types of Tags

feritedoc has notions about the types of tags. There are block level tags, grouping tags and attribute tags. If there is a better notation, please let me know.

3.2.1 Block Level Tags

Block level tags (such as @namespace) should be the first tag in a documentation block. They control what the documentation will apply to. Their scope is the entire comment.

Trying to include two block level tags in one comment will generally result in the following error message:

```
    Error:  Can't assign variables of type string and array
```

This won't cause feritedoc to abend, but the files that are output will be unusuable.

3.2.2 Grouping Tags

Grouping tags are a type of container. All tags between the start of the group and the end of the group are "in" the group. All the tags in a group are written to the same document file. The grouping tags are sorted alphabetically prior to being written to the document file.

The @end tag flags the end of a group.

Interestingly, some grouping tags can be nested. The html template uses a dot notation to indicate the container. For example, if @namespace bar is in @namespace foo, the index page will have an entry for both foo and foo.bar. That's not true for @group.

Gates of Madness, Next Right

feritedoc won't mind if you use the same name for multiple grouping tags, but you won't like the output. All documentation is written to the same output file, effectively losing most of the entries in each set. The index page generated will have the information for the first group processed, repeated once for each entry in the set. To top it off, all the links will bring up the information for the last group processed. Confusing and ugly. You've been warned.

3.2.3 Attribute Tags

An attribute tag specifies the value of an attribute. In a way, all tags are attribute tags. The @variable tag specifies the name of the variable. It's just that the block level and grouping tags have a larger impact.

Generally speaking, the value of an attribute tag will be all of the text between the tag and the next tag (or the end of the comment). Some tags (such as @param) will parse the text further. You would think that this is part of the parser, but it is actually part of the template. In html.fe, there is a function "generateTypeList" that seems to do this.

Using an attribute tag outside the scope of a block level tag will cause feritedoc to display the following error message:

```
Error:  Trying to access variable ’doctype’ in object ’b’ which is
null
```

This won’t cause feritedoc to abend, but the files that it outputs will be unusuable.

3.3 Markup Tags

The following tags are recognized by the parser.

- @brief

  A brief description of the item.

- @class

  The name of the class being documented. This is a block level, grouping tag.

- @declaration

  The signature of the function, with the names of any parameters.

- description

  A long description of the item.

- @example

  An example of using the item.

- @extends

  The name of the object being extended.

- @function

  The name of the function being documented. This is a block level, grouping tag.

- @group

  The name of the group being documented. This is a block level, grouping tag.

- @modifies

  The name of the object that is being modified.

- @module

  The name of the module being documented. This is a block level, grouping tag.

- @namespace

  The name of the namespace being documented. This is a block level, grouping tag.

- @param

  The parameter type and usage. In the form '@param type name description'.

- @return

  The type returned from the function. Can this apply to a module?

- @static

  Indicate that the item being documented is static.

- @type

  The type of the variable being documented.

- @variable

  The name of the variable being documented. This is a block level tag.

- @warning

  Document any warnings for the item.

- @protocol

  The name of the protocol being documented. This is a block level, grouping tag.

- @implements

  Document any protocols that a class implements. Used in the form '@implements protocol'.

4 Templates

To determine what templates are installed, go to /share/ferite/doc/template/. All .fe files in that directory with a class that extend Template are templates.

The templates in the current release of ferite include text and html.

4.1 Why create a template?

Insert intelligent text here.

4.2 How to create a template?

Insert intelligent text here.

5 Notes

Go ahead and take a look at the feritedoc script. You should find it in the bin directory of your distribution. You'll see a lot of system- specific setup and one important bit: "doc.fe." That indicates that feritedoc is actually a ferite script.

You may see older scripts with "* !class" instead of "* @class." Those scripts will not work with the current version of feritedoc.

How to make your template accept additional parameters.

I prefer using the –prefix to put documents into a separate directory.

6 Command Line Options

6.1 Overview

This section documents all of the feritedoc command line options.

6.2 Default Values

It is important to note that the feritedoc script will set the –ferite-prefix and –install-path options for you. You should only specify them if you need to override the default values.

6.3 Command Line Options

- –extra-file-name *scriptName*

  This command line option adds the script name specified to the list of scripts to extract documentation from. Unlike –file, this option can be specified multiple times and each script name will be documented.

  Please note that the scripts won't be documented until all other command line options have been parsed.

- –ferite-prefix *path*

  This command line option specifies the path to install ferite documentation to. This path will have "/lib/ferite/" appended to it.

  This option is only valid if the –regenerate option is also specified.

  It is interesting to contemplate what happens if both –prefix and –ferite-prefix are passed.

  The ferite prefix is automatically set by the feritedoc script. The value depends on your local installation. It should be the same as the output from running ferite-config –prefix.

- –file *scriptName*

  This command line option will place the script at the head of the list of scripts to extract documentation from. If there is already a script there, it will be replaced. You must use the –extra-file-name if you need to process multiple scripts.

  Please note that the script won't be documented until all other command line options have been parsed.

- –help

  This command line option causes feritedoc to print out a summary of the command line options. After doing so, the program exits.

- –install-path *value*

  This command line option tells feritedoc where to look to find the template files. feritedoc will append "/template/" to this path and

  The install path is automatically set by the feritedoc script. It defaults to the ferix prefix appended with "/share/ferite/doc."

- –prefix *value*

  This command line options tells feritedoc where to create the documentation files. If not specified, the files will be created in the current directory. If specified multiple times, only the last value will be used. The other values will be silently ignored.

  The prefix is prepended to every output file name. If you intend for the documentation files to go to a directory, you must ensure that the prefix ends with a "." character.

  The prefix defaults to the empty string if not specified.

- –regenerate

  This command line option causes feritedoc to create documentation for the ferite system. After doing so, the program exits. This should be the last option on the command line.

  This option forces the template type to "html."

  If you haven't specified a –prefix, then the prefix will be set to the ferite prefix, with "/share/doc/ferite/api/" appended to it.

- –template *templateName*

  This command line option specifies the name of the template file to use. This name should not include the ".fe" extension as feritedoc appends it automatically.

  There are currently two templates distributed with ferite. They are "text" and "html." These are both described in **templates** . If you specify an invalid template name, ferite will abend with the following error message.

  ```
  Unable to load template 'templateName', aborting
  ```

  If you don't supply a template name, ferite will abend with the following error message.

  ```
  No template specified - please specify one [eg.  text]
  ```

The template arguments are passed to the constructor during run_parser.

The template name defaults to the empty string if not specified.

Please note that the –regenerate option will override this value, forcing it to "html."

- –template-args *"commaSeparatedArgumentList"*

  This command line option causes feritedoc to pass the argument list to the template's constructor.

  The template arguments default to the empty string if not specified.

- –version

  This command line option causes feritedoc to print out the version number. After doing so, the program exits.